Lesson 3: Designing Progressive APIs

Krzysztof Cwalina
Program Manager

CLR Team

June 2004

Lesson 10: Designing Progressive APIs

- After successfully completing this lesson, you will be able to:
  - Design APIs that are both usable and powerful
  - Get to know the different developer personas
  - Understand the concept of "progressive" APIs

"Mort"

- Opportunistic Developer
- Microsoft® Visual Basic®, Web, domain expert
- Sense of achievement comes from creating solutions that "just work"
- Iterates on one problem at a time, learns by trial and error
- Uses components, maybe builds business objects, but not object-oriented (OO)
- Expects to program against physical objects, not abstract entities—File instead of Stream
- Focuses on the business problem, not the technology
- Most likely has no formal computer science education

"Elvis"

- Pragmatic Developer
- C#, majority of Microsoft developers
- Seeks a balance of power and productivity
- Likes to learn while doing
- Probably has a computer science degree or some formal CS training
- The ideal framework is OO from the ground up, maps logically to the domain, and has full functionality

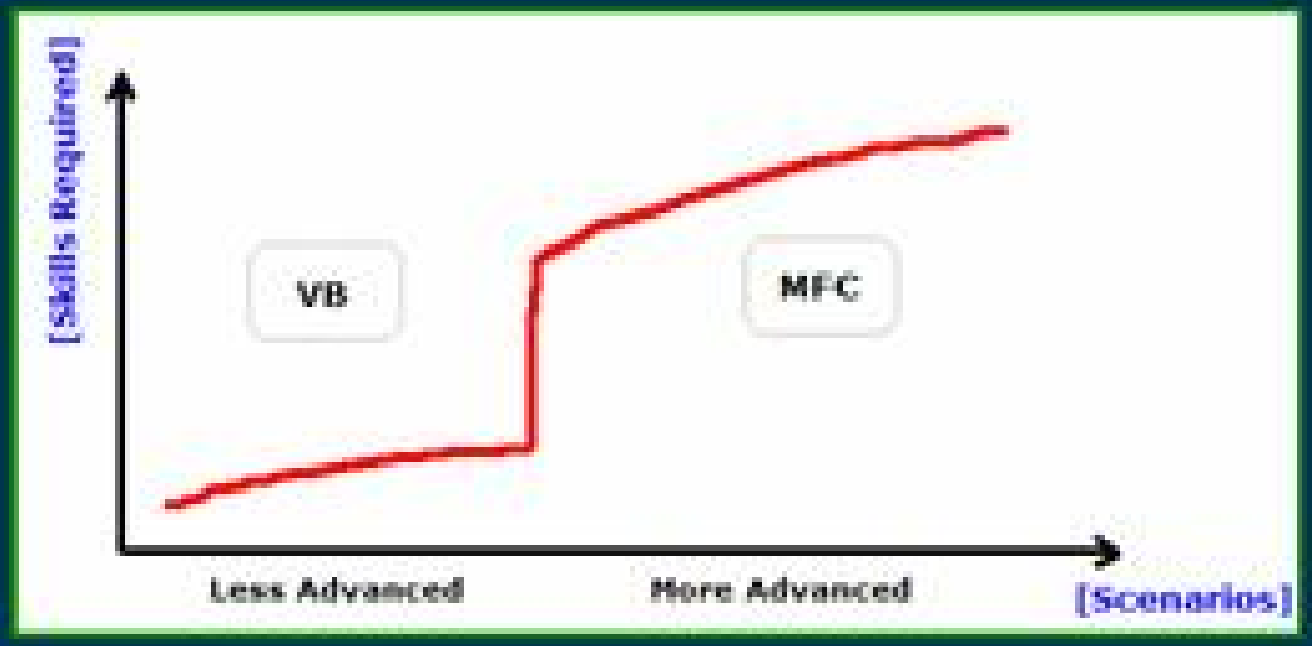
"Einstein"

- Paranoid Developer
- C++, system developer, academics
- Sense of achievement comes from creating efficient, high-performance code worthy of technical praise
- Learning style is proactive (likes to learn before doing)
- Wants to know he/she can replace any system components, thinks he/she can often do it better
- Doesn't like using black boxes

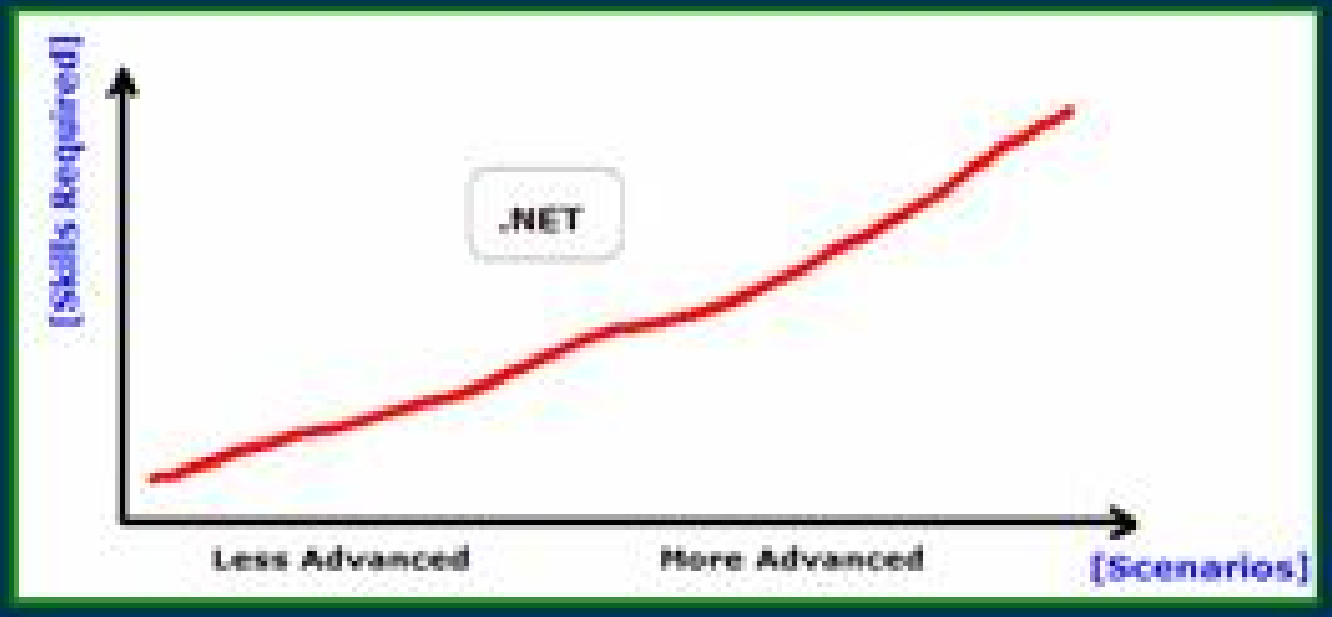
Progressive APIs

- Low barrier to entry
  - Easy to get started with
  - 80/20 Rule
  - Defaults and helpers
- Powerful
  - Richness
  - Performance
  - Scalability
- Consistent

Twentieth Century API Design



Twenty-First Century Progressive APIs



Principle: Scenario-Driven Design

- Define top scenarios
- Write code samples first, design API later
- Make top scenarios easy, make the rest possible
- Usability test top scenarios

Principles: Support Experimentation

- Obvious entry points
- Minimal initialization, sensible defaults
- Usage of properties
- Immediate feedback (exceptions)

Principles: Aggregate Component

- Component-oriented design
- Aggregate components (façades)
- Factored types
- Create, set, call usage pattern

```
MessageQueue q As New MessageQueue()

q.Path = "MyServer\OrdersQueue"
q.Formatter = New BinaryMessageFormatter()
q.Authenticate = true;

q.Send("Hello World")
```

Principles: Self-Documenting APIs

- APIs and documentation
- Domain correspondence
- Naming
- Consistency
- Exception messages
- Progressive documentation

Principle: Keeping Things Simple

- Number of objects
- Dependencies (configuration)
- Lines of code
- OO design methodologies and API design
- Abstractions

Lesson 3 Summary

- Mort, Elvis, Einstein
- Developers demand progressive APIs (usable and powerful)
- Scenario-driven design
- Support experimentation
- Aggregate components
- Self-documenting APIs
- Keep things simple